

# Programação Orientada a Objetos

## Aula Arquivo – Parte 2

### *Design Patterns e Arquivos*

### Turmas B e C

Nádia Félix  
([nadia.felix@ufg.br](mailto:nadia.felix@ufg.br))

Dirson S. Campos  
([dirson\\_campos@ufg.br](mailto:dirson_campos@ufg.br))

17/03/2022

# Programação Orientada a Objetos

## Classes Java File

- ❑ A classe Java File é um exemplo do padrão Composite da GoF para estruturação de Objetos.
- ❑ Composite Pattern (Objetos Composição) que é um padrão de projeto de software utilizado para representar um objeto formado pela composição de objetos similares. Este conjunto de objetos pressupõe uma mesma hierarquia de classes a que ele pertence.

# Programação Orientada a Objetos

## Classes Java File

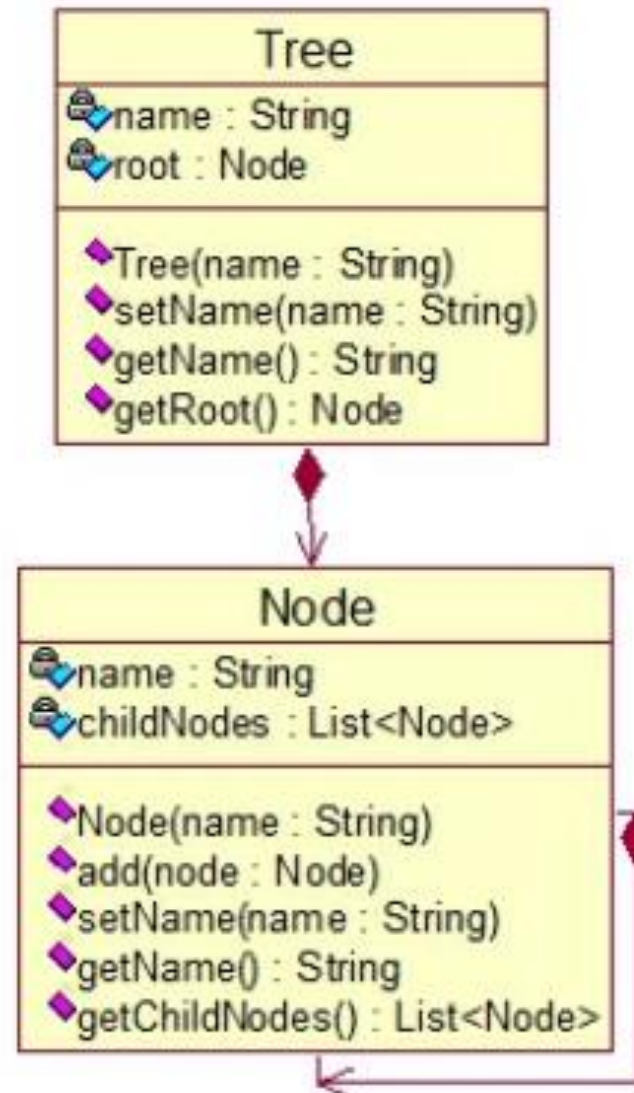
- ❑ O padrão Composite é, normalmente, utilizado para representar listas recorrentes ou recursivas de elementos como uma estrutura hierárquica e uma estrutura em árvore tal como a estrutura de Arquivo do Sistema Operacional que são manipuladas em Java pela classe File.

# Programação Orientada a Objetos

- ❑ Exemplo 1 – Representação de uma estrutura hierárquica de cidades dentro de países existentes no mundo.

# Programação Orientada a Objetos

## Exemplo 1 (UML)



# Programação Orientada a Objetos

## Exemplo 1 – Classe Tree

```
import java.util.*;
public class Tree {
    private Node root;
    protected String name;
    public Tree(String name) {
        this.root = new Node(name);
    }
    public Node getRoot() {
        return root;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

# Programação Orientada a Objetos

## Exemplo 1 – Classe Node

```
import java.util.*;
public class Node {
    protected String name;
    protected List<Node> childNodes;
    public Node(String name) {
        super(); this.name = name;
        this.childNodes = new ArrayList<Node>();
    }
    public void add(Node node) {
        childNodes.add(node); }
    public String getName() {
        return name;
    }
}
```

# Programação Orientada a Objetos

## Exemplo 1 – Classe Node

### (continuação)

```
public void setName(String name) {  
    this.name = name;  
}  
    public List<Node> getChildNodes() {  
        return childNodes;  
    }  
}
```



# Programação Orientada a Objetos

## Exemplo 1 – Classe TestaTree

```
import java.util.List;
public class TestaTree {
    public static void main(String[] args) {
        Tree tree=new Tree("Mundo");
        Node root = tree.getRoot();
        Node brasil = new Node("Brasil");
        Node america = new Node("America");
        root.add(brasil);
        root.add(america);
    }
}
```

# Programação Orientada a Objetos

## Exemplo 1 – Classe TestaTree (continuação)

```
Node sanfancisco = new Node("San Fancisco");  
Node newyork = new Node("New York");  
america.add(sanfancisco);  
america.add(newyork);  
System.out.println(root.getName());  
List<Node> childeNodes = root.getChildNodes();
```

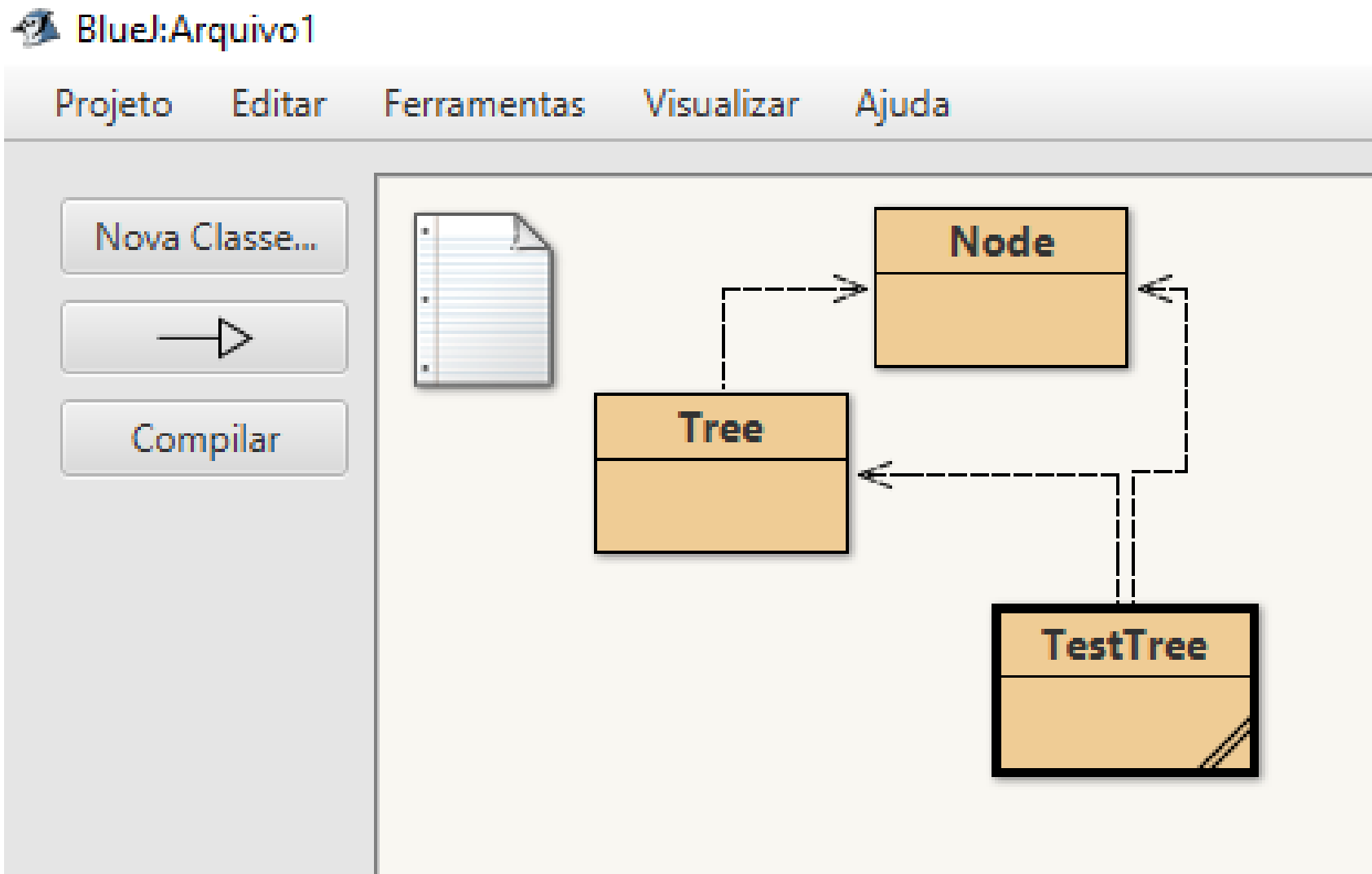
# Programação Orientada a Objetos

## Exemplo 1 – Classe TestaTree (continuação)

```
for (Node node : childeNodes) {  
    System.out.println("----" + node.getName());  
    List<Node> childNodes2 = node.getChildNodes();  
    for (Node node2 : childNodes2) {  
        System.out.println("-----" + node2.getName());  
    }  
}  
}
```

# Programação Orientada a Objetos

## Diagrama de Classes do Exemplo 1 no BlueJ



# Programação Orientada a Objetos

**Rodando Exemplo 1 no BlueJ e capturando a Janela de Execução.**

BlueJ:Arquivo1

Projeto    Editar    Ferramentas    Visualizar    Ajuda

Nova Classe...  
→  
Compilar

```
graph TD; Tree --> Node; TestTree --> Tree; TestTree --> Node;
```

BlueJ: BlueJ: Janela de Terminal - Arquivo1

Opções

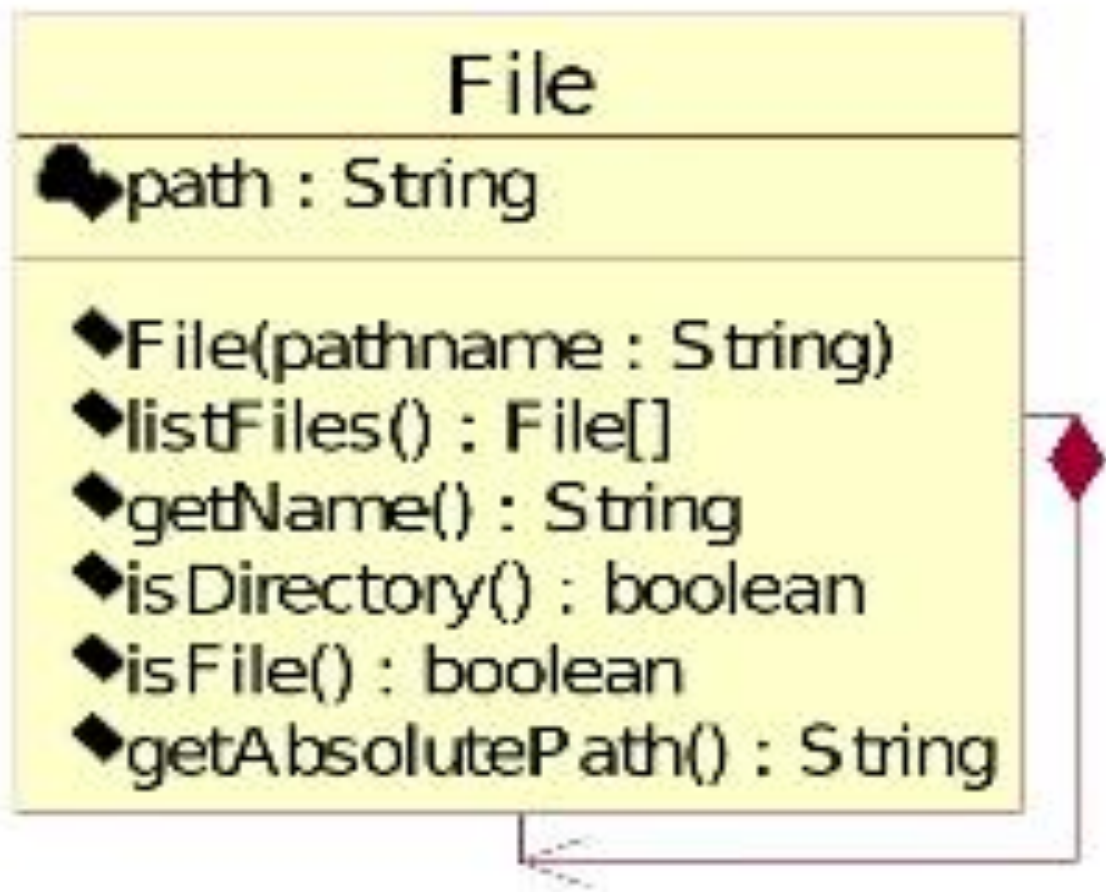
Mundo  
----Brasil  
-----Goiânia  
-----Anápolis  
----America  
-----San Francisco  
-----New York

# Programação Orientada a Objetos

- ❑ Exercício 1 – Adapte o código do **Padrão Composite** do Exemplo 1 para trabalhar com a Classe Java File mantendo as classes originais e criando uma nova classe de Teste.

# Programação Orientada a Objetos

## Exercício 1 (UML)



# Programação Orientada a Objetos

## Exercício 1 – Classe TestFile

(incompleta – falta o método main)

```
import java.io.File;
public class TestFulanoTS {
    private static String level = "";

    public static void main(String[] args) {

        //Coloque o seu código fonte aqui
        // De acordo com as instruções
    }
```



# Programação Orientada a Objetos

## Exercício 1 – Classe TestFile

(incompleta – falta o método main)

```
public static void showAllDirectory(String path)
{
    File dir = new File(path);
    // Lista todos os subdiretórios e
    // arquivos do diretório
    File[] dirs = dir.listFiles();
```

# Programação Orientada a Objetos

## Exercício 1 – Classe TestFile

(incompleta – falta o método main)

```
for (int i = 0; dirs != null && i < dirs.length; i++) {  
    File f = dirs[i];  
    if (f.isFile()) {  
        System.out.println(level + f.getName());  
    } else  
        if (f.isDirectory()) {  
            System.out.println(level + f.getName());  
            level += "----";  
            showAllDirectory(f.getAbsolutePath());  
            level = level.substring(0,  
                                    level.lastIndexOf("----"));  
        }  
    }  
}
```

# Programação Orientada a Objetos

- ❑ **Observação:** O código do Exercício 1 foi feito no mesmo Projeto e no mesmo pacote no BlueJ com a classe de teste acrescentada.
- ❑ O nome da classe acrescentada segue a seguinte regra:
  - O primeiro nome do estudante e as iniciais em maiúscula de seu sobrenome, por exemplo se o estudante chamar “**Fulano de Tal da Silva**” o nome da classe de teste é **TestFulanoTS.java**

# Programação Orientada a Objetos

❑ **Exercício 1 – Complete o método main da solução destes slides de modo que o diretório a ser mostrado é o diretório com o primeiro nome do Estudante e as iniciais de seu sobrenome.**

- Por exemplo se o estudante se chamar “**Fulano de Tal da Silva**” o nome do Diretório é **FulanoTS**.
- Dentro do Diretório deve ter um arquivo chamado **FulanoTS.zip**
- Dentro do Diretório criado deve ter outro diretório cujo o nome é a matrícula do estudante, se o matrícula do estudante for **202205033**
  - Dentro do diretório da matrícula deve ter dois arquivos de acordo com o nome do estudante, o primeiro no formato .doc e outro no formato .pdf. Se o se o estudante se chamar “**Fulano de Tal da Silva**” o nome dos arquivos são respectivamente **FulanoTS.doc e FulanoTS.pdf**.

# Programação Orientada a Objetos

## Diagrama de Classes do Exercício 1 do estudante

“Fulano de Tal da Silva”



Blue!Arquivo1

Projeto

Editar

Ferramentas

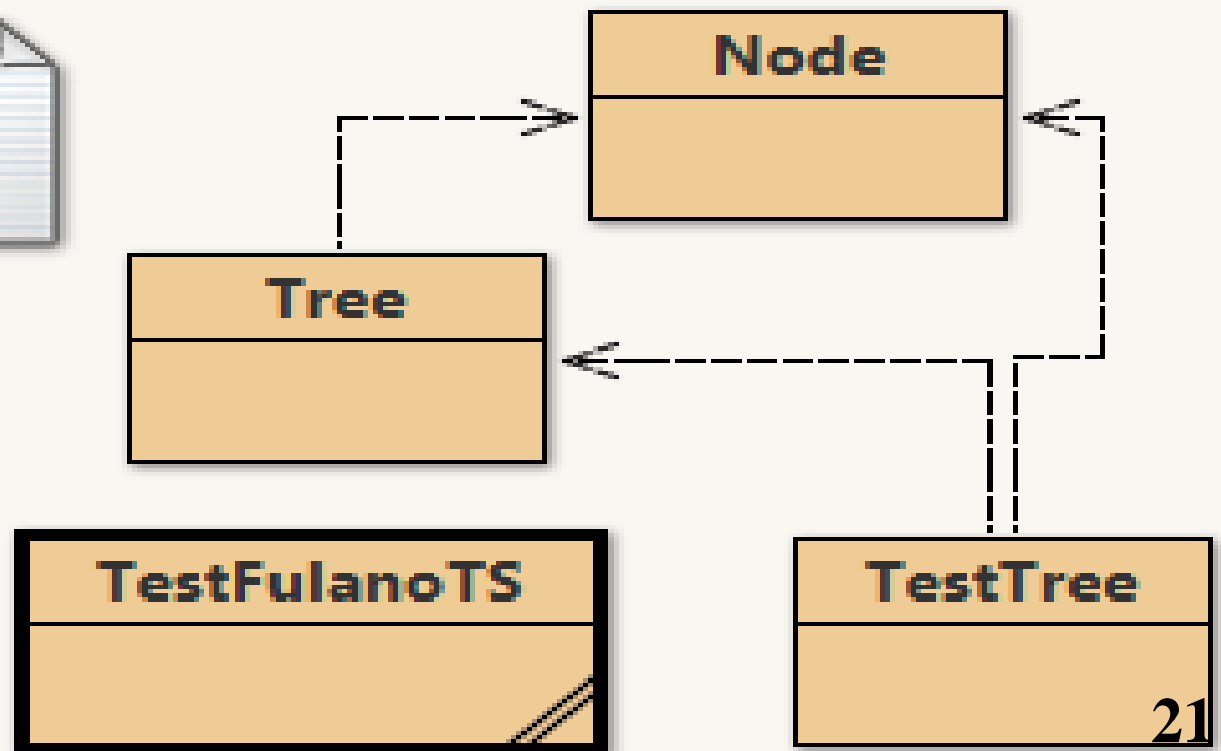
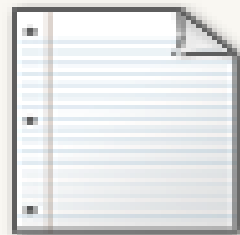
Visualizar

Ajuda

Nova Classe...

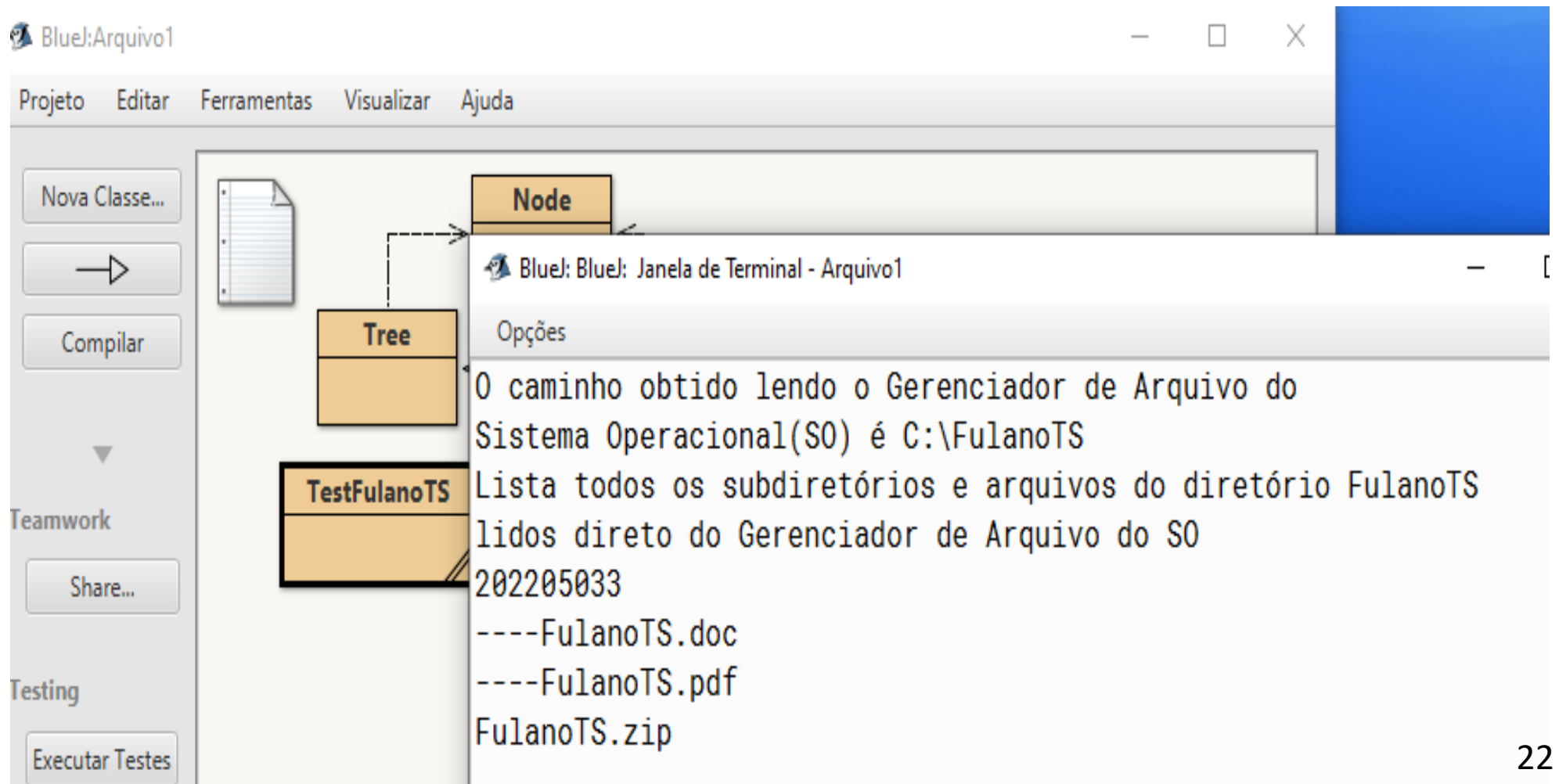


Compilar



# Programação Orientada a Objetos

## Rodando o Exercício 1 com os dados do estudante “Fulano de Tal da Silva”



The screenshot shows a Java IDE window titled "BlueJ:Arquivo1". The menu bar includes "Projeto", "Editar", "Ferramentas", "Visualizar", and "Ajuda". On the left, there are buttons for "Nova Classe...", a right-pointing arrow, "Compilar", and a "Teamwork" section with a "Share..." button. Below that is a "Testing" section with an "Executar Testes" button. The main workspace displays a class hierarchy with three classes: "Node", "Tree", and "TestFulanoTS". A dashed arrow points from "Tree" to "Node". The "TestFulanoTS" class is selected, and its code is visible in the editor. A terminal window titled "BlueJ: BlueJ: Janela de Terminal - Arquivo1" is open, showing the output of the program. The output text is as follows:

```
Opções
O caminho obtido lendo o Gerenciador de Arquivo do
Sistema Operacional(SO) é C:\FulanoTS
Lista todos os subdiretórios e arquivos do diretório FulanoTS
lidos direto do Gerenciador de Arquivo do SO
202205033
----FulanoTS.doc
----FulanoTS.pdf
FulanoTS.zip
```